

REMARKS

In the September 13, 2002, Office Action in this patent application, the United States Patent and Trademark Office (hereinafter the "Office") rejected Claims 1 and 3-19 under 35 U.S.C. § 103(a) as being unpatentable in view of the teachings of U.S. Patent No. 5,764,958, issued to Coskun (hereinafter "Coskun"), taken in view of the statements made in portions of this patent application (more particularly, statements made at pages 12-16), and more specifically, statements made at page 12, lines 21-25; page 13, lines 1-6; page 14, lines 24-25; and page 16, lines 13-17. In addition, Claim 2 was rejected under 35 U.S.C. § 103(a) as being unpatentable in view of the teachings of Coskun and statements made in portions of this application, taken in view of the teachings of U.S. Patent No. 5,867,708 issued to Copeland et al. (hereinafter "Copeland et al."). This amendment amends Claim 9 to remove extraneous commas so as to clarify the subject matter of the claimed invention.

Prior to discussing in detail why applicant believes that all of the claims in this application are allowable, a brief description of applicant's invention and a brief description of the teachings of the cited and applied references are provided. The following background and the discussions of the disclosed embodiments of applicant's invention and the teachings in the cited and applied references are not provided to define the scope or interpretation of any of the claims of this application. Instead, such discussions are provided to help the Office better appreciate important claim distinctions discussed thereafter.

Summary of Processes and Threads

Early operating systems allowed users to run only one program at a time. Users ran a program, waited for it to finish, and then ran another one. Modern operating systems allow users

LAW OFFICES OF
CHRISTENSEN O'CONNOR JOHNSON KINDNESS^{LLC}
1420 Fifth Avenue, Suite 2800
Seattle, Washington 98101
206.682.8100

to execute (run) more than one program at a time or even multiple copies of the same program at the same time. This change highlights a subtle distinction: the difference between a program and a process. A program is a static sequence of instructions whereas a process is the dynamic invocation of a program along with the system resources required for the program to run. Thus, a process, in the simplest terms, is an executing program.

Processes include one or more threads. A thread is the basic unit used by the operating system to allocate processor time. A thread can include any part of the process code, including parts currently being executed by another thread. A processor is capable of executing only one thread at a time. However, a multitasking operating system, i.e., an operating system that allows users to run multiple programs, appears to execute multiple programs at the same time. In reality, a multitasking operating system continually alternates between programs, executing a thread from one program, then a thread from another program, etc. As each thread finishes its subtask, the processor is given another thread to execute. The extraordinary speed of the processor provides the illusion that all of the threads execute at the same time. Multitasking increases the amount of work a system accomplishes because most programs do not require that threads continuously execute. For example, periodically, a thread stops executing and waits while a slow resource completes a data transfer or while another thread is using a resource it needs. When one thread must wait, multitasking allows another thread to execute, thus taking advantage of processor cycles that would otherwise be wasted.

While the terms multitasking and multiprocessing are sometimes used interchangeably, they have different meanings. Multiprocessing requires multiple processors. If a machine has only one processor, the operating system can multitask, but not multiprocess. If a machine has multiple processors, the operating system can both multitask and multiprocess.

In summary, an operating system having a process to run presumes the existence of a program from which the process is invoked because the program is the static aspect of the process and the process is the dynamic aspect of the program. Programs are typically created in an application development environment using an application development language. An application development environment is an integrated suite of applications for use by software developers to create programs. Typical components of application development environments include a compiler, a file browsing system, a debugger, and a text editor for use in creating programs. An application development language is a computer language designed for creating programs. The present invention as summarized below focuses on an application development environment, which in other words is a programming environment.

Summary of the Invention

Applicant's invention is directed to improving the development of programs so that programs may be developed in a more efficient and bug-free manner. As programs have become increasingly complex, programming environments are becoming a factor in the length of time it takes to create new programs and the number of bugs that occur as new programs are executed in conjunction with other programs. Traditionally, programs are developed in a linear manner. Unfortunately, a linear programming environment is often undesirable when highly complex computer programs that must run concurrently with other programs are to be developed. Another programming environment is the message-driven environment. In a message-driven programming environment, different objects interface with other objects via messages. Such messages typically are complex structures. To use such messages, the context must be unpacked from a message prior to the execution of an action. The message-driven programming environment as well as the linear programming environment are fragile and render development

of programs that must run concurrently with other programs potentially more difficult and bug-laden than is desired.

To solve or to reduce the foregoing problems, applicant's invention provides an asynchronous programming environment. More specifically, applicant's invention provides a dynamic object storage scheme for storing a plurality of objects, a dynamic dispatch scheme based on the plurality of objects, and an object recognition scheme to describe the plurality of objects. The dynamic object storage scheme stores the plurality of objects so that each object can be accessed as necessary by different threads within the asynchronous programming environment. The dynamic object storage scheme is dynamic in that objects can be created and removed as necessary during the execution of tasks within the asynchronous programming environment. The dynamic dispatch scheme is designed to invoke an action based on the plurality of objects stored by the dynamic object storage scheme. Each action that is invoked by the dynamic dispatch scheme falls into one of a plurality of categories. Actions may be classified differently at different times.

The plurality of categories include three categories—actions needing precisely one object, actions needing more than one object, or actions not needing an object at all. Actions needing precisely one object include message handlers, and other actions that do not need any object for their execution are generally used to create objects, such as default constructors and real-time routines. Actions that require multiple objects for their dispatch generally combine objects and perform tasks as designed by the programmer.

Other aspects of the present invention include an object recognition scheme that describes the plurality of objects stored by the dynamic object storage scheme. The description of objects allows the asynchronous programming environment to determine whether a described object fits the needs of an application programming interface. In other words, when an application

programming interface is presented to the asynchronous programming environment, it is not always clear which objects in a depository of objects fit the presented application programming interface. The object recognition scheme of the present invention allows one or more objects to be identified, thereby determining the suitability of one or more objects to the requirements of the application programming interface.

Unlike prior programming environments, an asynchronous programming environment allows thread-agnostic programs to be developed. Such programs have symmetric multithreading capabilities, i.e., they can be scaled to use the most of available processor processing power. Symmetric multithreading is expected to reduce instruction cache misses on hardware implementations that handle multiple instruction streams at once. An asynchronous programming environment is more efficient than a message-driven programming environment because execution is driven via presence of objects so that little or no translation is needed to recover context of execution. These attributes of an asynchronous programming environment allow programs to be developed in a more efficient and more bug-free manner.

Summary of Coskun

Coskun is directed to a method to allow an object (as in an object-oriented programming environment) to have different roles. In a sociological environment, a person has one or more roles, such as student, teacher, father, etc. To simulate the sociological environment using an object-oriented programming environment, a person could be represented by a person class; a student could be represented by a student class; a teacher could be represented by a teacher class; and a father could be represented by a father class. But consider a situation where a person (an entity) can be both student and father (roles) in the sociological environment. To simulate this situation (where an entity can have multiple roles), object-oriented programming environments provide the mechanism of multiple inheritance, which allows a new class to be derived from

several existing classes. For example, to imitate an entity that is both a person, student, and father, a new class can be designed to inherit multiply from the person class, the student class, and the father class so that objects instantiated from the new class can have different responsibilities depending on different contexts. The problem, according to Coskun, is that the multiple inheritance mechanism causes inherited objects to carry the complexity overhead of multiple classes in every context. In other words, when an object is acting as a father, it does not make sense for the object to carry the overhead associated with the student class. The solution proposed by Coskun is to have a mechanism to dynamically add roles to objects depending on the context of the object while eliminating the overhead requirements by jettisoning the use of multiple inheritance.

Summary of Copeland et al.

Copeland et al. is directed to a system for automatically inserting concurrency services into classes of an information handling system employing an object-oriented programming environment. The insertion of concurrency services is accomplished by using a multiple inheritance mechanism. The system of Copeland et al. provides steps for recognizing that an object does not support concurrency and generating a concurrent version of the object. There are several alternatives for implementing the step of generating a concurrent version of an object but all of the alternatives involve the use of multiple inheritance.

One alternative for generating a version of an object supporting concurrency is called automatic transactional locking. This approach includes the step of inheriting from a Lockable class, which adds state to a class of the object to allow object-level locking. The approach further includes the step of inheriting from a Serialized method class, which adds serializing methods to all existing methods of the non-concurrent object. Furthermore, the approach further

includes designing the class of the non-concurrent object to inherit from a RecoverableObject class.

Yet another alternative for implementing the act of generating a version of an object supporting concurrency is called automatic per method locking. This approach includes a step for inheriting from a Lockable class, which adds state to a class, and a step of inheriting from a Serialized metaclass, which adds serializing methods to all methods of the non-concurrent object.

Yet a further alternative for implementing the act of generating a version of an object supporting concurrency is called explicit locking. This approach includes inheriting from a concurrent object class ConcurrentObject, which itself is derived from the Lockable class. Moreover, this approach includes inheriting from a RecoverableObjects class.

The Claims Distinguished

As discussed in greater detail below, the claims of the present application are clearly patentably distinguishable over the teachings of the above-cited references. The present invention is directed to provide an asynchronous programming environment, which solves or reduces problems associated with prior programming environments. Applicant's invention provides a dynamic object storage scheme for storing a set of objects, a dynamic dispatch scheme based on the set of objects, and an object recognition scheme to describe the set of objects. The dynamic object storage scheme stores the set of objects so that each object can be accessed as necessary by different threads within the asynchronous programming environment. The dynamic object storage scheme is dynamic in that objects can be created and removed as necessary during the execution of tasks within the asynchronous programming environment. The dynamic dispatch scheme is designed to invoke an action based on the plurality of objects stored by the dynamic objects storage scheme. The action that is invoked by the dynamic dispatch scheme falls into one of a plurality of categories. Action may be classified differently at

different times. The plurality of categories of actions include three categories--actions needing precisely one object, actions needing more than one object, or actions not needing any object at all. Actions needing precisely one object include message handlers, and other actions that do not need any objects for the execution are generally used to create objects, such as default constructors and real-time routines. Actions that require multiple objects for the dispatch generally combine objects and perform tasks as designed by the developer. The object recognition scheme describes the set of objects stored by the dynamic objects storage scheme. The description of objects allows the asynchronous programming environment to determine whether a described object fits the needs of an application programming interface. In other words, when an application programming interface is presented to the asynchronous programming environment, it is not always clear which objects in a depository of objects fit the presented application programming interface. The object recognition scheme of the present invention allows one or more objects to be identified, thereby determining the suitability of one or more objects to the requirements of the application programming interface.

Regarding the claims, independent Claim 1 is directed to an asynchronous programming environment. The environment is recited as comprising a dynamic object storage scheme for storing a plurality of objects. The environment further comprises a dynamic dispatch scheme for invoking an action that belongs to one of a plurality of categories. The plurality of categories includes needing one object, needing more than one object, and needing no object. The environment yet further comprises an object recognition scheme for providing a description of each object of the plurality of objects. The description allows a determination of whether an object described by the description fits an application programming interface.

Claims 2-4 are dependent from independent Claim 1 and are directed to further limitations of the environment described above. Claim 2 is dependent on Claim 1 and recites

that in the plurality of objects as stored via the dynamic object storage scheme are accessible utilizing a recyclable locking mechanism. Claim 3 is dependent on Claim 1 and recites that the plurality of objects as described via the object recognition scheme, each comprises a series of tokens, and each token in turn relates to an attribute of the object. Claim 4 is dependent on Claim 1 and recites that the dynamic dispatch scheme provides for execution of objects based on unpacked-into-messages events.

Independent Claim 5 is directed to a method, and the method is recited as comprising storing a plurality of objects via a dynamic object storage scheme. The method further comprises dispatching at least one of the plurality of objects via a dynamic dispatch scheme based on events from at least one of the plurality of objects. The dynamic dispatch scheme is capable of invoking an action that belongs to one of a plurality of categories. The pluralities of categories include needing one object, needing more than one object, and needing no object. The method yet further comprises describing each of the plurality of objects utilizing an object recognition scheme. The object recognition scheme provides a description of each object of the plurality of objects. The description allows a determination of whether the object described by the description fits an application programming interface. Claims 6-8 are dependent from independent Claim 5 and are directed to further limitations of the method described above. Claim 6 is dependent on Claim 5 and recites that the act of storing a plurality of objects via a dynamic objects storage scheme comprises accessing one of the plurality of objects utilizing a recyclable locking mechanism. Claim 7 is dependent on Claim 5 and recites that the act of describing each of the plurality of objects utilizing an object recognition scheme comprises describing each of the plurality of objects as a series of tokens. Each token relates to an attribute of the object. Claim 8 is dependent on Claim 5 and recites that the act of dispatching at least one

of the plurality of objects via a dynamic dispatch scheme comprises executing at least one of the plurality of objects based on unpacked-into-messages events.

Independent Claim 9 is directed to a computer, and the computer is recited as comprising a processor; a computer-readable medium; and an asynchronous programming environment being executed by the processor from the medium. The environment is recited by independent Claim 9 as comprising a dynamic object storage scheme for storing a plurality of objects. The environment is further recited as comprising a dynamic dispatch scheme based on an event from at least one of the plurality of objects for invoking an action that belongs to one of a plurality of categories, the plurality of categories including needing on object, needing more than one object, and needing no object. The environment yet further comprises an object recognition scheme for providing a description of each object of the plurality of objects. The description allows a determination of whether an object described by the description fits an application programming interface.

Claims 10-13 are dependent from independent Claim 9 and are directed to further limitations of the computer described above. Claim 10 is dependent on Claim 9 and recites that the plurality of objects as stored via the dynamic object storage scheme are accessible utilizing a recyclable locking mechanism. Claim 11 is dependent on Claim 9 and recites that in the plurality of objects as described via the object recognition scheme, each comprises a series of tokens, and each token in turn relates to an attribute of the object. Claim 12 is dependent on Claim 9 and recites that the dynamic dispatch scheme provides for the execution of objects based on unpacked-into-messages events. Claim 13 is dependent on Claim 9 and recites that the medium comprises a piece of memory.

Independent Claim 14 is directed to a computer-readable medium that has a computer program stored thereon for execution on a computer. The computer program is recited by

independent Claim 14 as providing an asynchronous programming environment. The environment comprises a dynamic object storage scheme for storing a plurality of objects. The environment further recites a dynamic dispatch scheme based on events from at least one of the plurality of objects for invoking an action that belongs to one of a plurality of categories, the plurality of categories including needing one object, needing more than one object, and needing no object. The environment yet further recites an object recognition scheme for providing a description of each object of the plurality of objects. The description allows a determination of whether an object described by the description fits an application programming interface.

Claims 15-19 are dependent from independent Claim 14 and are directed to further limitations of the computer-readable medium described above. Claim 15 is dependent on Claim 14 and recites that the plurality of objects as stored via the dynamic object storage scheme are accessible utilizing a recyclable locking mechanism. Claim 16 is dependent on Claim 14 and recites that in the plurality of objects as described via the object recognition scheme, each comprises a series of tokens, and each token in turn relates to an attribute of the object. Claim 17 is dependent on Claim 14 and recites that the dynamic dispatch scheme provides for execution of objects based on unpacked-into-messages events. Claim 18 is dependent on Claim 14 and recites that the medium comprises a Compact Disc Read-Only Memory (CD-ROM). Claim 19 is dependent on Claim 14 and recites that the medium comprises a floppy disk.

As noted above, the final Office Action rejected Claims 1 and 3-19 under 35 U.S.C. § 103(a) as being unpatentable in view of the teachings of Coskun taken in view of statements made in portions of this patent application. Claim 2 was rejected under 35 U.S.C. § 103(a) as being unpatentable in view of the teachings of Coskun and statements made in portions of this patent application taken in view of the teachings of Copeland et al. As also noted above,

applicant respectfully disagrees. The cited and applied references simply fail to teach all of the limitations of the independent claims, much less the recitations of many of the dependent claims.

Focusing on Claim 1, there is no teaching or suggestion in the cited references for an asynchronous programming environment in the manner recited in Claim 1. Claim 1 recites that the environment comprises a dynamic object storage scheme for storing a plurality of objects. The remaining recitations of Claim 1 are directed to a dynamic dispatch scheme for invoking an action that belongs to one of a plurality of categories. The plurality of categories are recited by Claim 1 to include needing one object, needing more than one object, and needing no object. Claim 1 also recites an object recognition scheme for providing a description of each object of the plurality of objects. The description allows a determination of whether an object described by the description fits an application programming interface. The cited and applied references neither teach a dynamic dispatch scheme nor an object recognition scheme, among other things.

Coskun describes a mechanism to add roles to objects in an object-oriented programming environment. Roles can be dynamically added to objects. These objects are called dynamic objects by Coskun. Each dynamic object is a list of objects. In other words, each dynamic object is basically a container object that contains other objects. Different objects can be added to or subtracted from the list at any time to change the role of a dynamic object. For example, when a program is started, a person object has a student object representing the capability of the person object (which in this case is a list or container of roles) representing capabilities or behaviors of a student. Before calling a method requiring teacher characteristics, a teacher role (or a teacher object) is added to the person object. Once the method has completed its execution and returns to the main programming flow, the teacher role is deleted or subtracted from the person object. Coskun's technique of adding and subtracting roles from an object is designed to

eliminate the need to use multiple inheritance of prior object-oriented programming environments.

Contrarily, Copeland et al. requires the use of multiple inheritance to insert concurrency services into a class for an information handling system that employs an object-oriented programming environment. The method of Copeland et al. for inserting concurrency services into a class includes the steps of recognizing that an object does not support concurrency and generating a concurrent version of the object. Copeland et al. describes several alternatives for generating concurrent versions of the object. But all of the alternatives demand as necessary or essential the use of multiple inheritance. For example, one alternative for generating a version of an object supporting concurrency is called automatic transactional locking. This approach not only inherits from a Lockable class, which adds states to allow object-level locking, but also inherits from both a Serialized metaclass, which adds serialization to the object, and a RecoverableObject class. Another alternative is described by Copeland et al. for generating a version of an object supporting concurrency. This alternative is called automatic per method locking. Like the first approach, this approach also includes inheriting from multiple classes, such as the Lockable class and the Serialized metaclass. A third alternative method is described by Copeland et al. for generating a version of an object supporting concurrency, and this alternative is called explicit locking. Similarly, to the above-described alternatives, the third approach not only requires inheriting from a RecoverableObjects class but also a ConcurrentObject class (which itself is derived or inherited from the Lockable class). See the Abstract of Copeland et al. Figures 4 and 5 of Copeland et al. are especially revealing. Note the inheriting tree of a class called "Dog". The Dog' class inherits from at least six other classes, namely, Dog', Lockable, RecoverableObject, Serialized, Recoverable, and SOMMBeforeAfter.

Whereas Coskun is focused on eliminating the use of multiple inheritance, Copeland et al. instead requires the use of multiple inheritance. To combine, either the approach of Coskun must be abandoned or the use of multiple inheritance by Copeland et al. must be jettisoned, and a combination would destroy the operation of the references. Thus, no *prima facie* case of obviousness has been established.

Even if Coskun were combinable with Copeland et al. (which applicant specifically denies), the combination could not teach applicant's invention. For example, Claim 1 recites "an object recognition scheme for providing a description of each object of the plurality of objects, the description allowing a determination of whether an object described by the description fits an application programming interface." The Office has insisted (in the final Office Action) that Coskun teaches the claimed "object recognition scheme" at column 3, lines 21-25. This cannot be correct. The portions of Coskun cited by the Office describe a class, which in object-oriented programming is a generalized category that is used in a program to define a set of attributes or a set of services that characterize any member of the class. For argument purposes, applicant agrees that a class contains information that is descriptively relevant to the instantiation of all objects in a class. However, the distinction between a class and an object must be noted to understand applicant's invention. Information in a class is available during the time of compilation but is extinguished once executable code is produced from the compilation process. An object can exist only during run time, which is the time period during which a compiled program is running. The recited "object recognition scheme" provides a description of each object during run time. That is not the case with the portions of Coskun cited by the Office.

The Office has proposed to combine Coskun with statements made by applicant's patent application at page 14, lines 24-25. At the cited portions, applicant incorporates by reference a reference by Ondrej Such, Applications of Stochastic Asynchronous Programming Technique to

LAW OFFICES OF
CHRISTENSEN O'CONNOR JOHNSON KINDNESS^{LLC}
1420 Fifth Avenue, Suite 2800
Seattle, Washington 98101
206.682.8100

Procedure Testing (May 1998). The Office should note that the author of this cited reference (Ondrej Such) is the identical inventive entity of the present application. According to MPEP § 2132, the term "others" in 35 U.S.C. § 102(a) refers to any entity which is different from the inventive entity. Furthermore, according to MPEP § 2132, this holds true for all types of references eligible as prior art under 35 U.S.C. § 102(a) including publications. Because Ondrej Such is the inventor of the present patent application as well as the author of the incorporated reference, the incorporated reference is not a proper reference to combine with Coskun for a rejection under 35 U.S.C. § 103(a). Thus, the rejections in the final Office Action are improper because each rejection is entirely dependent on the combination of Coskun and the incorporated reference by Ondrej Such.

The Office cited Copeland et al. for the teaching of the claimed "recyclable locking mechanism." This also cannot be correct. The Office explained that the recyclable locking mechanism is shown by the "locking and unlocking ability of the lock manager" of Copeland et al. What makes applicant's recyclable locking mechanism recyclable is that there exists a pool of a limited number of locks. When the lock is no longer necessary, the lock returns to the pool so that it can be reused, and hence is recyclable. In the case of Copeland et al., in order for an object to obtain concurrency services, the object is required to carry the overhead of a lock. To put it simply, each object in Copeland et al. must have its own lock in order to participate in concurrency processes. The problem with Copeland et al. is that the overhead of a lock in each object would rob precious computing resources in order to provide the maintenance of a lock. No lock is recyclable in the system of Copeland et al. because each object needs its own lock. Not so with applicant's invention. Given the defect of Copeland et al., there would be no benefit to combining Coskun and the incorporated reference by Ondrej Such, with Copeland et al. Even

if the combination were possible, which assumption applicant specifically denies, the combination still would not teach applicant's invention.

Clearly, neither Coskun nor Copeland et al., alone much less in combination, teaches or suggests the subject matter of Claim 1. More specifically, none of these references, alone much less in combination, teaches or suggests an asynchronous programming environment that has a dynamic dispatch scheme, which invokes an action that belongs to one of a plurality of categories (needing one object, needing more than one object, and needing no object), and an object recognition scheme, which provides a description of each object of the plurality of objects so as to allow a determination of whether an object described by the description fits an application programming interface, as recited in Claim 1.

As will be readily appreciated in the foregoing discussion, none of the cited and applied references teaches or suggests the subject matter of Claim 1. Specifically, none of the cited and applied references teaches an object recognition scheme in the manner recited in Claim 1. As a result, applicant submits that Claim 1 is clearly allowable in view of the teachings of the references.

With respect to dependent Claims 2-4, all of which depend directly or indirectly from Claim 1, it is clear that the subject matter of these claims is also not taught or suggested by the cited and applied references, namely Coskun or Copeland et al. Claims 2-4 all add limitations that are clearly not taught or suggested by any of the cited and applied references, particularly when the limitations are considered in combination with the recitations of the claims from which these claims individually depend. In summary, Claims 2-4 are submitted to be allowable for reasons in addition to the reasons why Claim 1 is submitted to be allowable.

Independent Claim 5 is directed to a method, which recites an act for storing a plurality of objects via a dynamic object storage scheme. The method further recites an act for

dispatching at least one of the plurality of objects via a dynamic dispatch scheme based on events from at least one of the plurality of objects. The dynamic dispatch scheme is capable of invoking an action that belongs to one of a plurality of categories. The plurality of categories includes needing one object, needing more than one object, and needing no object. The method of Claim 5 yet further recites an act for describing each of the plurality of objects utilizing an object recognition scheme. The object recognition scheme provides a description of each object of the plurality of objects. The description allows a determination of whether an object described by the description fits an application programming interface. For generally the same reasons discussed above with respect to Claim 1, applicant submits that the subject matter of Claim 5 is not taught or suggested by any of the cited and applied references, and thus, that Claim 5 is also allowable.

With respect to dependent Claims 6-8, all of which depend directly or indirectly from Claim 5, it is clear that the subject matter of these claims is also not taught or suggested by the cited and applied references, namely, Coskun or Copeland et al. Claims 6-8 all add limitations that are clearly not taught or suggested by any of the cited and applied references, particularly when the limitations are considered in combination with these recitations of the claims from which these claims individually depend. In summary, Claims 6-8 are submitted to be allowable for reasons in addition to the reason why Claim 5 is submitted to be allowable.

Independent Claim 9 is directed to a computer. The computer recites a processor, a computer-readable medium, and an asynchronous programming environment being executed by the processor from the medium. The computer of Claim 9 further recites that the environment comprises a dynamic object storage scheme for storing a plurality of objects. The environment yet further recites a dynamic dispatch scheme based on events from at least one of the plurality of objects for invoking an action that belongs to one of a plurality of categories. The plurality of

categories includes needing one object, needing more than one object, and needing no object. The environment additionally recites an object recognition scheme for providing a description of each object of the plurality of objects. The description allows a determination of whether an object described by the description fits an application programming interface. For generally the same reasons discussed above with respect to Claims 1 or 5, applicant submits that the subject matter of Claim 9 is not taught or suggested by any of the cited and applied references, and thus, that Claim 9 is also allowable.

With respect to dependent Claims 10-13, all of which depend directly or indirectly from Claim 9, it is clear that the subject matter of these claims is also not taught or suggested by the cited and applied references, namely, Coskun or Copeland et al. Claims 10-13 all add limitations that are clearly not taught or suggested by any of the cited and applied references, particularly when the limitations are considered in combination with the recitations of the claims from which these claims individually depend. In summary, Claims 10-13 are submitted to be allowable for reasons in addition to the reasons why Claim 9 is submitted to be allowable.

Independent Claim 14 is directed to a computer-readable medium that has a computer program stored thereon for execution on a computer. The computer program is recited by Claim 14 as providing an asynchronous programming environment. This environment comprises a dynamic object storage scheme for storing a plurality of objects. The environment is further recited to include a dynamic dispatch scheme based on events from at least one of the plurality of objects for invoking an action that belongs to one of a plurality of categories. The plurality of categories includes needing one object, needing more than one object, and needing no objects. The environment as recited by Claim 14 further includes an object recognition scheme for providing a description of each object of the plurality of objects. The description allows a determination of whether an object described by the description fits an application programming

interface. For generally the same reasons discussed above with respect to Claims 1, 5, or 9, applicant submits that the subject matter of Claim 14 is not taught or suggested by any of the cited and applied references, and thus, that Claim 14 is also allowable.

With respect to dependent Claims 15-19, all of which depend directly or indirectly from Claim 14, it is clear that the subject matter of these claims is also not taught or suggested by the cited and applied references, namely Coskun or Copeland et al. Claims 15-19 all add limitations that are clearly not taught or suggested by any of the cited and applied references, particularly when the limitations are considered in combinations with the recitations of the claims from which these claims individually depend. In summary, Claims 15-19 are submitted to be allowable for reasons in addition to the reasons why Claim 14 is submitted to be allowable.

In light of the foregoing remarks, it is clear that none of the cited and applied references teaches, let alone renders unpatentable, Claims 1-19. The cited and applied references are all directed to adding roles to an object without using multiple inheritance or using multiple inheritance to add concurrency services to objects. The present invention is directed to an entirely different concept and solution. The present application is directed to providing an asynchronous programming environment so that programs may be developed in a more efficient and bug-free manner.

Allegedly Admitted Prior Art

The Office has alleged that various portions of applicant's disclosure are admitted prior art. Applicant respectfully traverses such an allegation. For example, the Office has indicated that applicant's disclosure teaches that a recyclable locking mechanism is prior art. Applicant respectfully disagrees. No reasonable interpretation of the language of applicant's disclosure provides support for the conclusion that a recyclable locking mechanism is prior art. The recyclable locking mechanism referenced in the present application is the subject of another

copending patent application, which is not prior art. While the present application indicates that a dynamic object storage scheme is known within the art and that threads are known within the art, there is no express or implied admission that a recyclable locking mechanism is so known. Accordingly, applicant respectfully requests that the Office withdraw these allegations.

Request to Correct Inventorship Under M.P.E.P. § 605.04(g)

The Office is hereby notified of a typographical or transliteration error in the spelling of the inventor's name. The incorrect inventor's name of "Slovak Ondrej Such" should read "Ondrej Such." In the originally-submitted declaration of December 21, 1998, the name of the inventor in the present patent application is set forth as "Ondrej Such." The inventor's citizenship was incorrectly set forth as "Czech Republic." The incorrect citizenship designation "Czech" was crossed out and the correct citizenship designation "Slovak" was handwritten so as to rectify the incorrect citizenship designation. The Office incorrectly interpreted, as evidenced by the filing receipt received on January 28, 1999, that the name of the inventor is "Slovak Ondrej Such." Applicant respectfully requests that this incorrect name designation be changed to the correct name designation of "Ondrej Such." More particularly, applicant respectfully requests that the Examiner have the Technology Center's technical support staff enter the correct name designation "Ondrej Such" in the PALM database or other pertinent databases and print out a new bibliographic data sheet, which should be placed in the file wrapper of the present patent application.

CONCLUSION

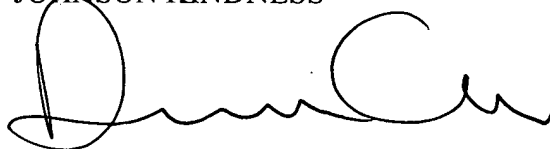
In view of the foregoing remarks, applicant submits that all of the claims in the present application are clearly patentably distinguishable over the teachings of Coskun and Copeland et al. Thus, applicant submits that this application is in condition for allowance. Reconsideration and re-examination of the application and allowance of the claims and passing

LAW OFFICES OF
CHRISTENSEN O'CONNOR JOHNSON KINDNESS^{PLLC}
1420 Fifth Avenue, Suite 2800
Seattle, Washington 98101
206.682.8100

of the application to issue at an early date are solicited. If the Examiner has any remaining questions concerning this application, the Examiner is invited to contact the applicant's undersigned attorney at the number below.

Respectfully submitted,

CHRISTENSEN O'CONNOR
JOHNSON KINDNESS^{PLLC}



D.C. Peter Chu
Registration No. 41,676
Direct Dial No. 206.695.1636

I hereby certify that this correspondence is being deposited with the U.S. Postal Service in a sealed envelope as first class mail with postage thereon fully prepaid and addressed to the U.S. Patent and Trademark Office, P.O. Box 2327, Arlington, VA 22202, on the below date.

Date: November 12, 2002 Lindy A. Weston

DPC:jeh

In the Claims:

9. (Twice Amended) A computer comprising:

- a processor;
- a computer-readable medium; and
- an asynchronous programming environment executed by the processor from the medium,

the environment comprising:

- a dynamic object storage scheme for storing a plurality of objects;
- a dynamic dispatch scheme based on events from at least one of the plurality of objects[,] for invoking an action that belongs to one of a plurality of categories, the plurality of categories including needing one object, needing more than one object, and needing no object; and
- an object recognition scheme[,] for providing a description of each object of the plurality of objects, the description allowing a determination of whether an object described by the description fits an application programming interface.